Proc. Monterey Workshop: Radical Innovations of Software and Systems Eng. in the Future, US Army Research Office (Venice, Italy, Oct. 2002).

System-of-Systems Design From An Object-Oriented Paradigm

Dale Scott Caffall
Technical Director, Test and Assessment
Missile Defense Agency
7100 Defense Pentagon
Washington, D.C. 20301-7100
Butch.Caffall@mda.osd.mil
(703) 697-4556

Dr. J. Bret Michael
Associate Professor, Comp. Science Dept.
Naval Postgraduate School, Code CS
833 Dyer Road
Monterey, CA 93943-5118
bmichael@nps.navy.mil
(831) 656-2655

Abstract. Safety critical interactions increase as the complexity of highly integrated systems increases. In complex systems and systems-of-systems, these possible combinations are practically limitless. System "unravelings" have an intelligence of their own as they expose hidden connections, neutralize redundancies, bypass firewalls, and exploit chance circumstances for which no system engineer might plan. A software fault in one module of the system software may coincide with the software fault of an entirely different module of the system software. This unforeseeable combination can cause cascading failures within the system.

In this paper, we will offer a new paradigm for system-of-systems development using a hypothetical missile defense system as a case study. Rather than decompose each system within the missile defense system in the traditional functional fashion, we will treat the system-of-systems as a single entity – a system – that is comprised of various abstract classes. We will identify the common problems and shortcomings that system engineers address in the design and development of system-of-systems.

The object-oriented paradigm offers a new system-of-systems requirements and design methodology that can minimize accidental complexity and control essential complexity through the object-oriented concepts of decentralized control flow, minimal messaging between classes, implicit case analysis, and information-hiding mechanisms.

The greatest source of system software faults will occur in the integration of the various systems. With respect to our case study, the hypothetical missile defense systems will be a complex product that will contain many discrete software packages within each system. As a rule, these software packages will be developed independent of each other and programmed in many different languages. Additionally, the hypothetical missile defense system will include legacy systems that are currently in operation. The means of integrating these elements and legacy systems are intricate tactical data links that support the message transfer within the system-of-systems.

While the hypothetical missile defense system will not be a pure object-oriented design, we can incorporate many of the principles of object-oriented technology to decrease the complexity of the system-of-systems. We believe that software engineers of system-of-systems can use this object-oriented paradigm to produce a sound design for the system-of-systems rather than the traditional federation of systems through a highly coupled communication medium.

Proc. Monterey Workshop: Radical Innovations of Software and Systems Eng. in the Future, US Army Research Office (Venice, Italy, Oct. 2002).

Introduction. During the past decade, systems-of-systems have exploded into the battlespace of the joint and coalition warfighters. The acquisition community's response to the rabid craving for more accurate information and more lethal functionality has been a less than stellar hobbling of various legacy systems and ongoing system developments through tightly coupled and lowly cohesive communication shackles.

While there are many issues with system-of-systems acquisitions, the first issue that we must address is the requirements definition and allocation issue. Just as the requirements issue continues to plague single system acquisitions, the requirements issue is much more complicated in the system-of-systems acquisition. For example, a good number of the systems that comprise the system-of-systems are legacy systems and currently operate as stand-alone capabilities in the operational world. We developed these legacy systems with specific sets of requirements and with specific system functionality in mind. Additionally, just as we developed the legacy systems, we are developing new systems that will become a member of a system-of-systems under similar conditions. That is, we are developing these systems as stand-alone capabilities with specific sets of requirements and with specific system functionality in mind.

Now comes the desire to slam these various systems together and connect these systems through some communication medium in the hope of achieving greater functionality (i.e. the whole will be greater than the sum of its parts). We identify the systems that will form the system-of-systems, and we set out to bend, fold, spindle, and mutilate these systems in the fevered hope of producing a functional composition. Oftentimes, it is very difficult to think about the system-of-systems as a single entity so we mistakenly focus on modifying individual systems with little deliberation and consideration for the whole.

Our tools for reasoning about a system-of-systems typically consist of little more than a "sticks and circles" diagram. The "circles" represent the various systems that comprise the system-of-systems while the "sticks" are means of information transfer, a messaging protocol, and, perhaps, a translator box to translate the messaging format from one system to another. Armed with this sophomoric view of the system-of-systems, we attempt to analyze and describe the system-of-systems through a trivial picture of the various systems as connected by a convoluted labyrinth of lines.

Traditionally, this methodology failed to achieve an interoperable and integrated system-of-systems. With each new failure, the system engineers attempted to "tighten up" the protocol standard; however, the system-of-systems did not achieve the desired degree of interoperability and integration. The end-state is a collection of systems that are tightly coupled with a realized protocol standard that only serves to greatly increase the system-of-systems software complexity.

As we have witnessed time and again, system software critical interactions increase as the complexity of highly integrated systems increases. In the complex system-of-systems, these possible combinations are practically limitless. System "unravelings" have an intelligence of their own as they expose hidden connections, neutralize redundancies, bypass firewalls, and exploit chance circumstances for which no system engineer might

plan. [1] A software fault in one module of the system software may coincide with the software fault of an entirely different module of the system software. This unforeseeable combination can cause cascading failures within the system.

How do we reason about such a structure so that we have at least a modicum of chance to realize a functional system-of-systems? Can we extend the existing set of tools that we use in reasoning about a single system development to the more complex system-of-systems development? If true, can we use these tools to identify potential sources of accidental system software complexity?

We propose that applying the Unified Modeling Language (UML) and object oriented design (OOD) techniques to the system-of-systems requirements analysis offers a new model for reasoning about complex system-of-systems developments. Rather than disparate reasoning about the individual systems of a proposed system-of-systems, we propose that we develop a sound model for reasoning about the system-of-systems as a single, functional entity.

We will propose this new paradigm through an example of a hypothetical missile defense system. As depicted below in Figure 1, missile defense systems are most often analyzed and described by the sticks and circles diagram.

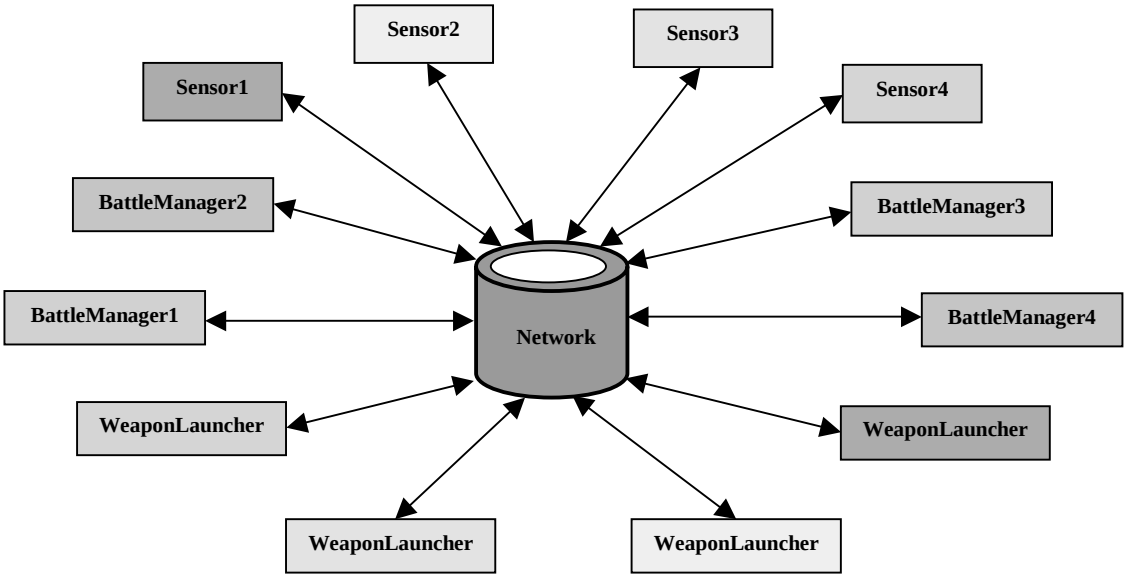

Figure 1. Hypothetical Missile Defense System-of-Systems

It is difficult to reason about requirements and analyze the system-of-systems with the hypothetical missile defense system-of-systems sticks and circles view. Although presented as a single entity, it is challenging to understand the affect of requirements changes and component limitations in this view. As previously mentioned, our reasoning

tendency is to focus on the individual systems of the system-of-systems in the hope that the desired functionality wondrously appears.

Unfortunately, magic and marvel are not tools that are abundantly available to system developers. Their fervent yet futile hopes for integrated systems and desired functionality too often fall shattered on the road of broken acquisition dreams. Frustration and antipathy are the frequent products of the most system-of-systems development.

Let us propose another view of the hypothetical missile defense system in which we apply UML and OOD techniques. We will develop a class diagram with abstract classes for the major components of the system-of-systems. We will reason about the class diagram in our attempt to develop subclasses to which we can begin to allocate requirements and analyze system capabilities and limitations. Additionally, we will identify message requirements and message flow in our attempt to reduce coupling in the system-of-systems by developing requirements for simplified interfaces between the components. Finally, we will propose a reassignment of methods to increase the cohesion of the components.

The first step is to develop a class diagram of abstract classes. For the hypothetical missile defense system-of-systems, we will use the following five classes:

- **Threat Missile:** The threat missile class is the enemy missile that contains warhead of mass destruction: nuclear, chemical, or high explosive munitions. The adversary will launch the threat missile within the confines of his state. The missile will climb into the exo-atmospheric region that constitutes up to 80% of the missile flight. The missile will re-enter the atmosphere over our forces or defended assets at which time it will impact at its aim point.

- **Sensor:** The sensor class is the object that detects the threat missile. Sensor is an abstraction of two subclasses: infrared class and radar class.

- **BM/C2:** The Battle Manager/Command and Control (BM/C2) class processes track data from the sensor. The BM/C2 monitors the threat missile, develops firing solution to negate the threat missile, and directs a weapon to launch its interceptor with the BM/C2-provided firing solution. The BM/C2 class is an abstraction for all system echelons of battle management.

- **Weapon:** The weapon class develops firing solutions, calculates the probability of kill, and implements the BM/C2 authorization to engage the threat missile.

- **Interceptor:** The interceptor class is the engagement mechanism that negates the threat missile. The interceptor class is the abstraction for both directed and kinetic energy intercepts of the threat missile.

Proc. Monterey Workshop: Radical Innovations of Software and Systems Eng. in the Future, US Army Research Office (Venice, Italy, Oct. 2002).

Given these classes and associated definitions, we constructed a class diagram of the hypothetical missile defense system as depicted below in Figure 2.

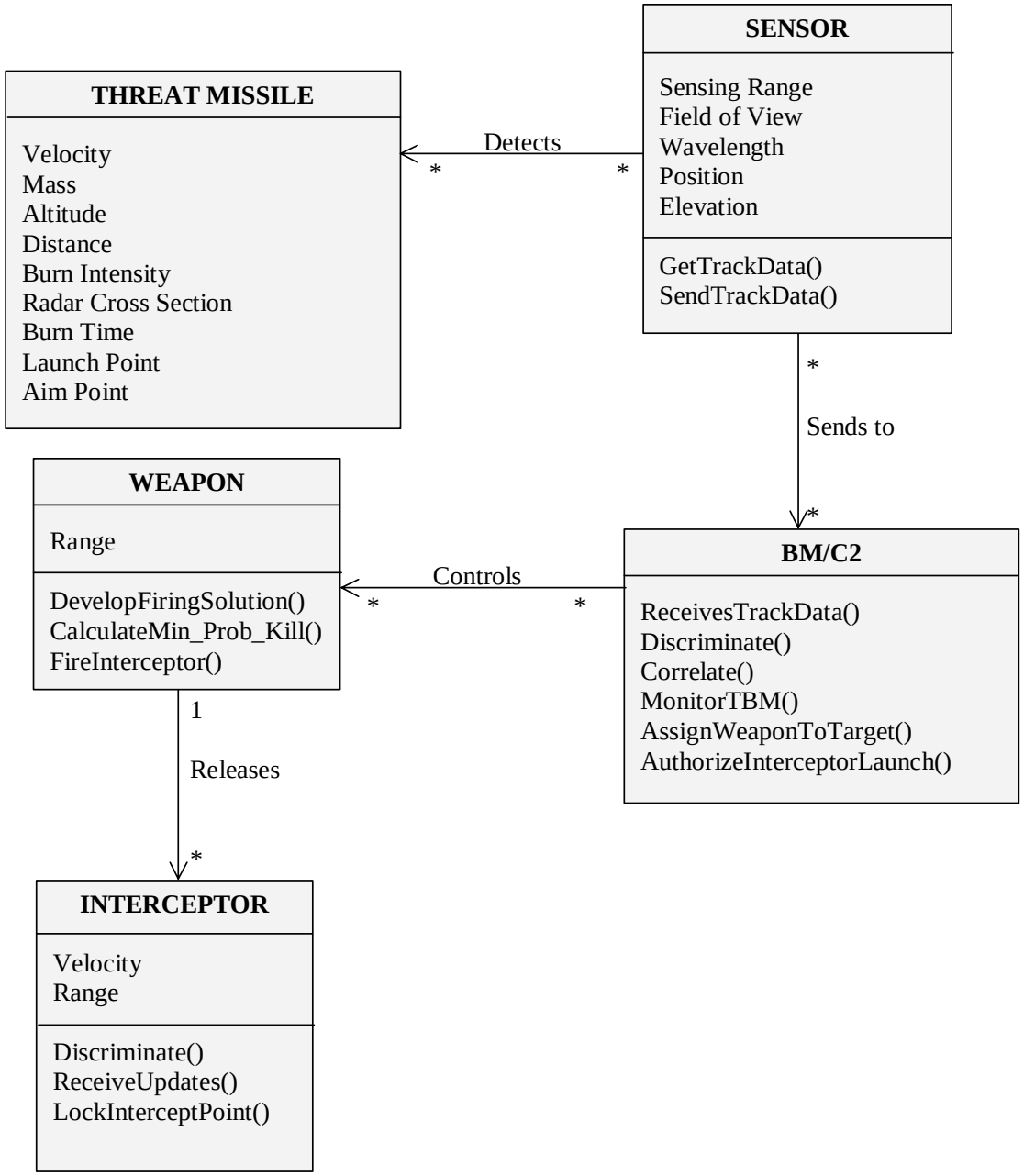

Figure 2. Class Diagram of Hypothetical Missile Defense System-of-Systems

Note that the message requirements in the above class diagram are very specific as compared to the single, large network interface of the sticks and circles diagram. Through this class diagram, we can easily determine the messaging requirements of each class. For example, the sensor class wants to determine the attributes of the threat missile

class. The BM/C2 class wants formed track data from the sensor class. The weapon class waits for control data from the BM/C2 class. The interceptor class waits for the interceptor release command from the weapon class.

From this class diagram, we can begin to define abstract interfaces between the classes. Rather than the largely unmanageable and complex network interface of the sticks and circles diagram, we can begin to develop very specific interface requirements from the class diagram approach.

Let us add detail to the threat missile class as this is the point of reference for our hypothetical missile defense system. We can develop subclasses (i.e. short range, intermediate range, and long range threat missiles) of the threat missile class as depicted below in Figure 3.

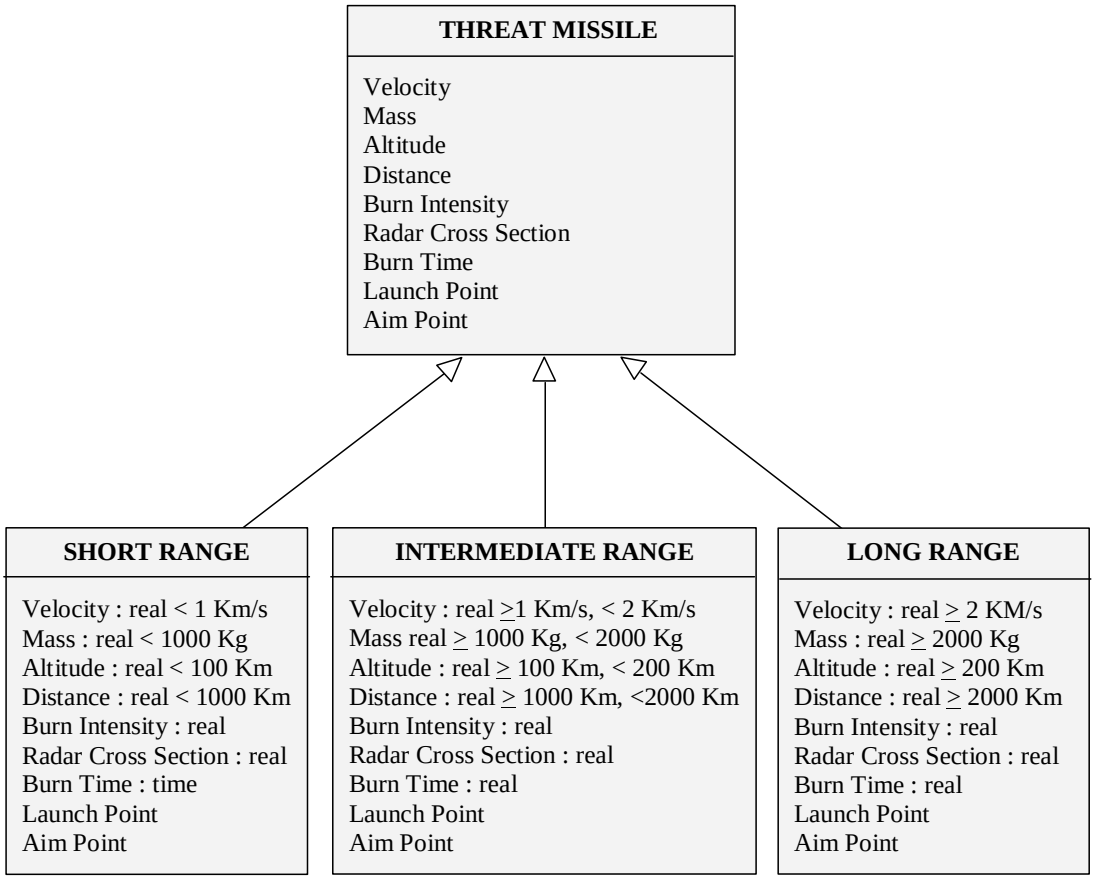

Figure 3. Subclasses of Threat Missile Class*

*Note: All attribute values listed in subclasses are fictitious and do not represent real threat missile data.

Proc. Monterey Workshop: Radical Innovations of Software and Systems Eng. in the Future, US Army Research Office (Venice, Italy, Oct. 2002).

In our definition of the subclasses, we have assigned attribute values. In our example, we have assigned fictitious data so that our example remains out of the classified regime. These subclasses with the assigned attributes will form the basis for our reasoning about the hypothetical missile defense system.

The sensor class is responsible for detecting the threat missile class so let us develop subclasses that can detect the threat missile subclasses that we have defined. The subclasses for the sensor class are depicted below in Figure 4.

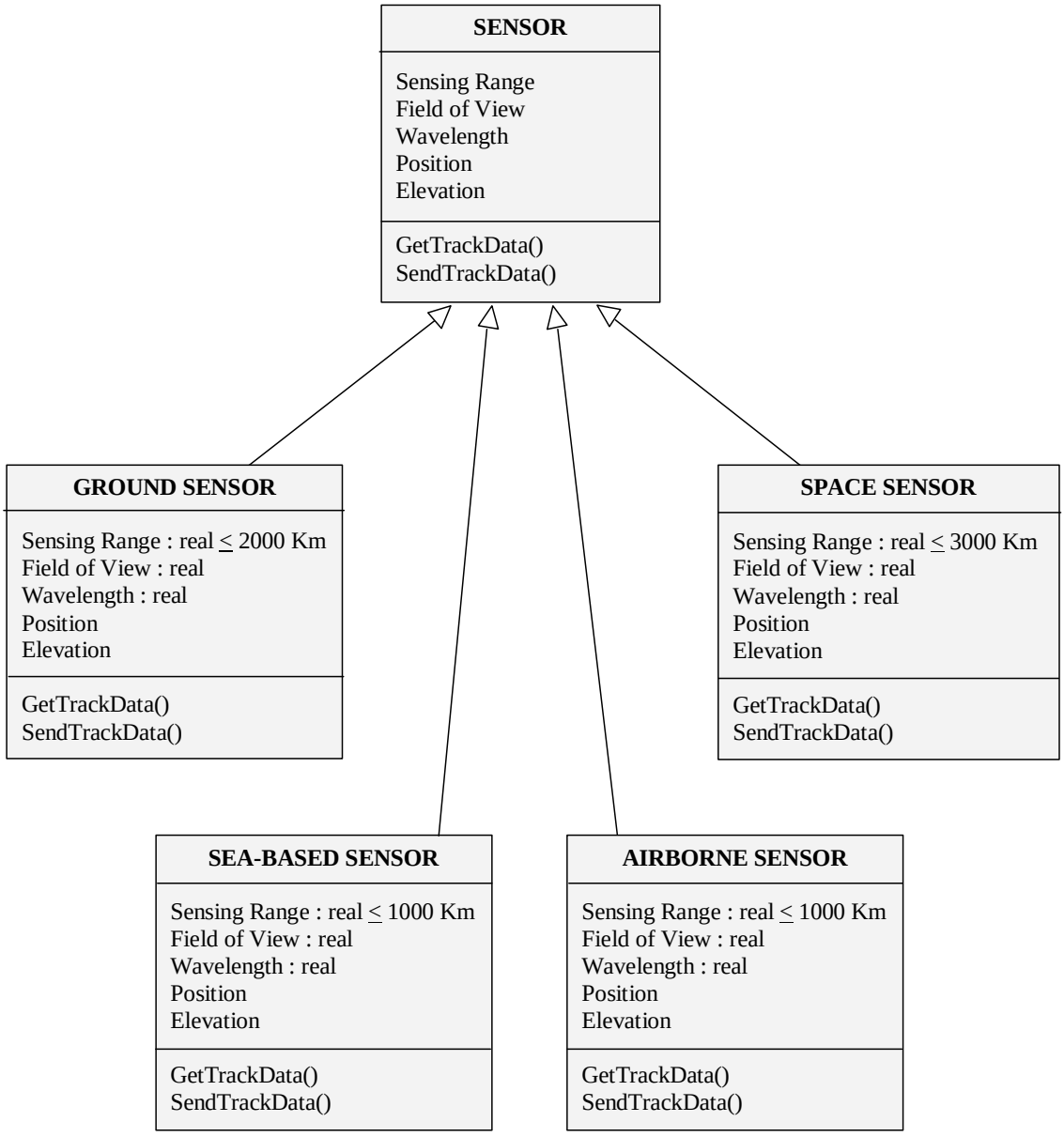

Figure 4. Subclasses of Sensor Class*

*Note: All attribute values listed in subclasses are fictitious and do not represent real sensor data.

Proc. Monterey Workshop: Radical Innovations of Software and Systems Eng. in the Future, US Army Research Office (Venice, Italy, Oct. 2002).

By considering the subclasses of the threat missile class, we can design a sensor framework for which we can attain overlapping coverage of our sensor subclasses to greatly increase our opportunities for the detection of the threat missiles. Additionally, we can develop additional requirements to bolster our detection capability. For example, after considering the threat missile subclasses for a potential adversary, we may desire to increase the sensing range of the Sea-Based Sensor to extend our coverage into an adversary's territory into which a Ground Sensor solution is not feasible. We can now levy this requirement change on the Sea-Based Sensor subclass.

After we have detected a Threat Missile object, then we must develop a firing solution and engage the threat missile. As depicted in Figure 2, the BM/C2 class handles these functions and several other important functions. While these functions are related, the incorporation of these methods in a single class lessens the cohesion of the class. Rather than a single BM/C2 class, we might develop the BM/C2 class as an aggregate of several classes as depicted below in Figure 5.

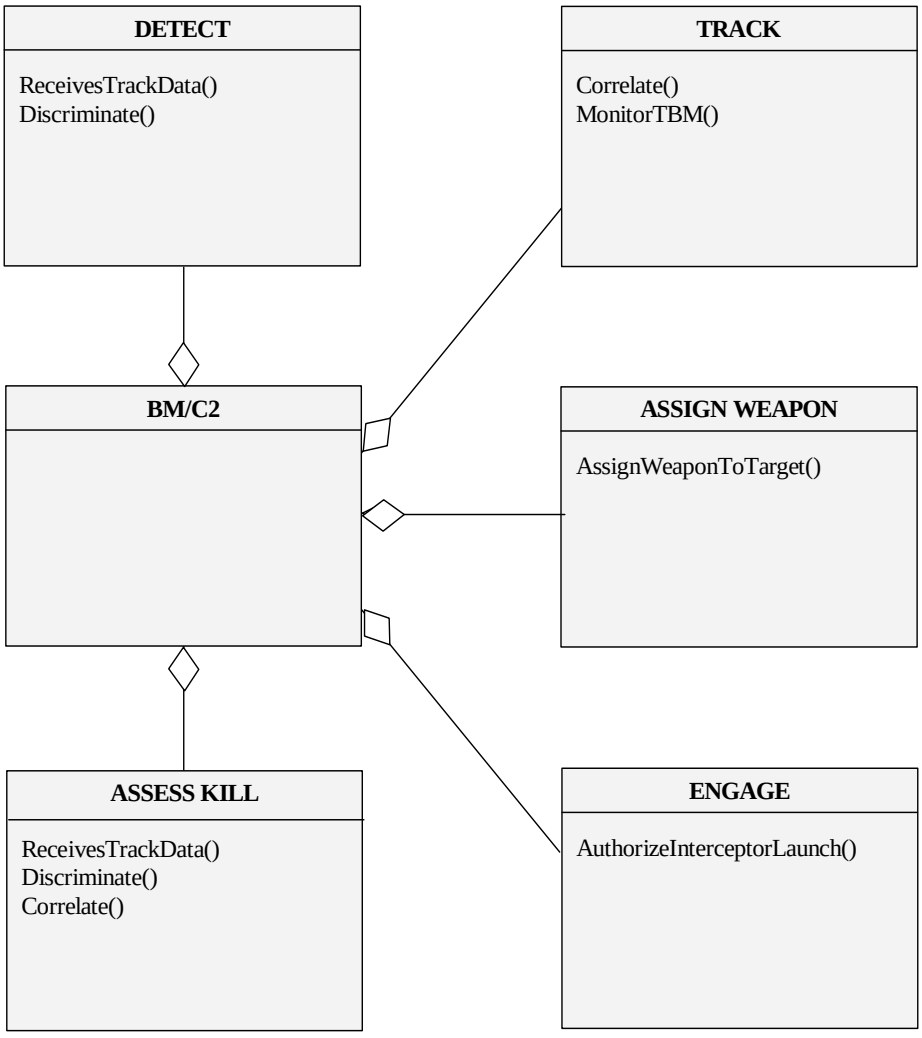

Figure 5. BM/C2 Class as an Aggregate

Proc. Monterey Workshop: Radical Innovations of Software and Systems Eng. in the Future, US Army Research Office (Venice, Italy, Oct. 2002).

As depicted in Figure 2, we separated the methods for developing and realizing a firing solution from the BM/C2 class and assigned these methods to the Weapon class. These methods are similar in function so the cohesion of this class is high. This separation is important as the realizations of the BM/C2 class and the Weapon class may physically reside on different hardware platforms. So, in addition to increasing the cohesion, we reduce the coupling by substituting more interfaces that are small and better defined for the larger interface required for data flow and messaging of the sticks and circles architecture depicted in Figure 2. The Weapon class and subclasses are show below in Figure 6.

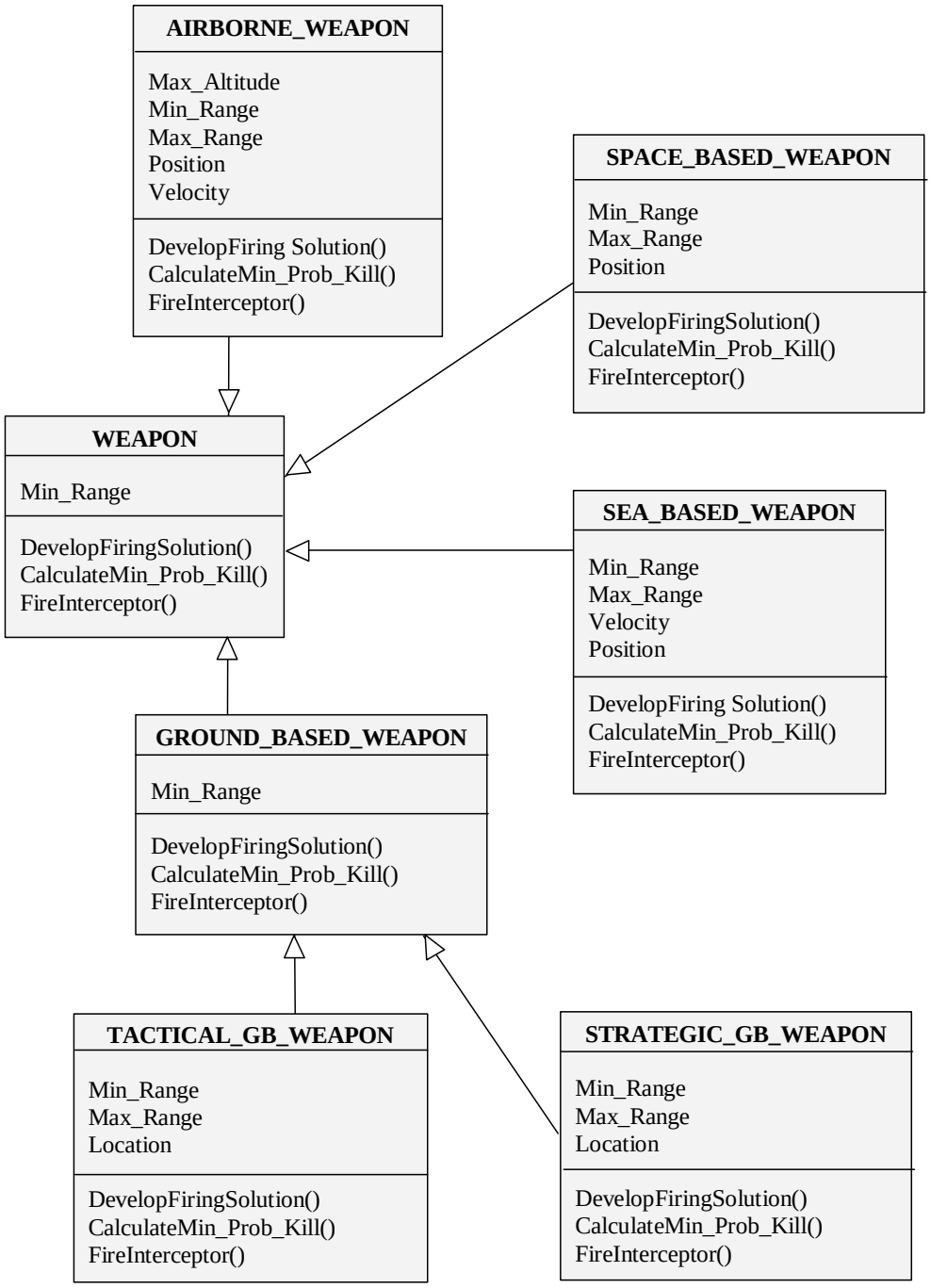

Figure 6. Subclasses of Weapon Class

Proc. Monterey Workshop: Radical Innovations of Software and Systems Eng. in the Future, US Army Research Office (Venice, Italy, Oct. 2002).

Finally, we consider the Interceptor class. Given the attributes of the Threat Missile class as well as potential deployment of our hypothetical missile defense system, we can develop the attributes and associated requirements for the Interceptor class. For example, the velocity of the Intermediate Range subclass of the Threat Missile class ranges between 1 Km/second and 2 Km/second and the distance of this same subclass ranges from 1000 Km to 2000 Km. As we consider the minimum altitude in which we must negate the threat missile to ensure minimal ground effects of the resulting debris, we can determine minimum velocities for our three subclasses of the Interceptor class. These subclasses are depicted below in Figure 7.

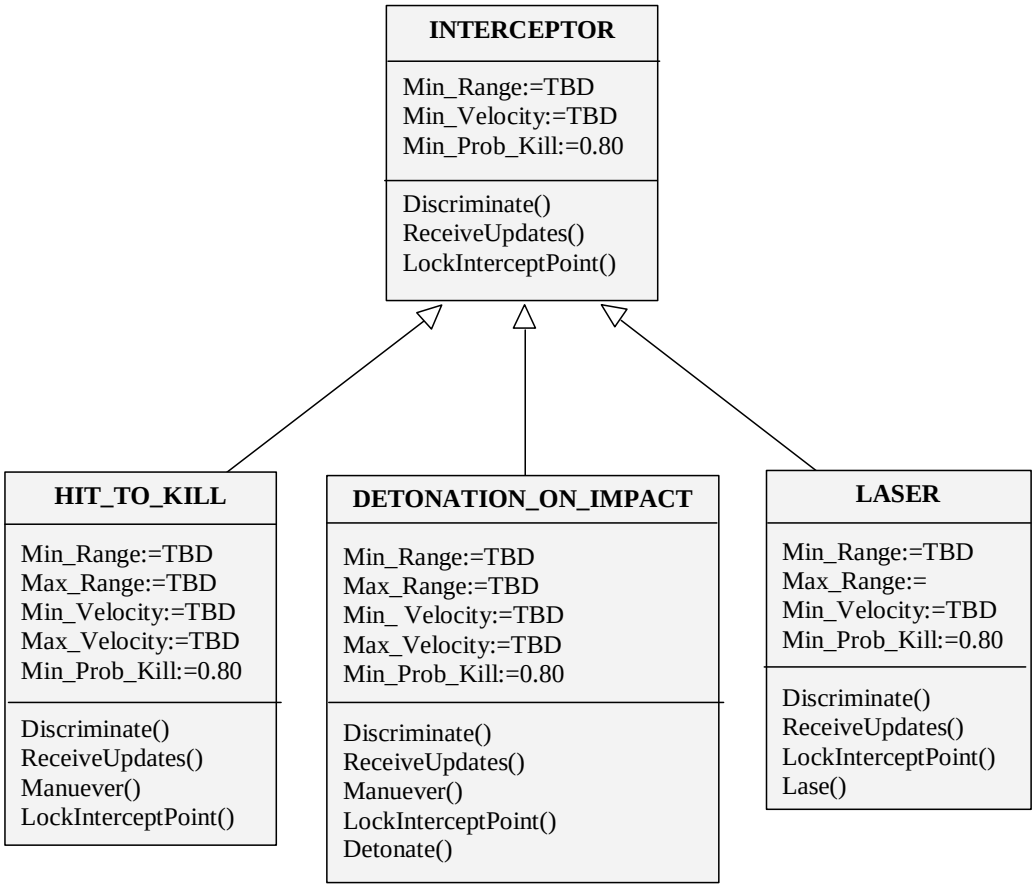

Figure 7. Subclasses of Interceptor Class

As we reason about the classes and subclasses of our hypothetical missile defense system, we can see that we will develop many interfaces in the realization that replaces the single, large network interface of the sticks and circles diagram of Figure 2. This is important to us in that we can manage a larger number of small, well-defined interfaces; however, the single, large network interface is much too unwieldy and complicated to manage effectively. We can reduce the messaging requirements of the large network interface to only that which is necessary for realizing the subclasses of our system-of-systems. Because the interface requirements are now manageable and known to all the system

developers, we have enhanced our ability to effectively integrate these systems into a system-of-systems.

As we define the class and subclass attributes, the concept of inheritance becomes important in that the allocation of requirements through attributes and methods ensures consistency in the realization of the subclasses in our developments. Each system developer will know the minimum set of requirements that must be implemented and each developer knows what requirements the other developers will realize.

By careful assignment of methods to each class, we can avoid the creation of the so-called "god class" that performs the bulk of the work within the system-of-systems. [2] Typically, we overload the battle manager function with the vast majority of the work. More often than not, the battle manager software contains many dissimilar tasks and requires a complex messaging network. Rather than primarily exchanging control or triggering messages among several classes, the typical battle manager requires the continual transport of great amounts of data that results in more complex rules of messaging and bandwidth requirements. By employing the aforementioned UML and OOD techniques, we can reassign methods to other classes in which these methods are better suited.

For example, consider the discriminate method listed in the BM/C2 class in Figure 2. This requires that the Sensor class send a great deal of data to the BM/C2 class. Perhaps we might reason that the Sensor class should contain the discriminate method and send a much smaller, refined track file to the BM/C2 class for prosecution. This would greatly reduce the messaging requirements and greatly simplify the interface between the Sensor class and the BM/C2 class.

As we reason about the classes and subclasses of the hypothetical missile defense system, we find that we can modify the methods to maximize the benefits of data hiding within the appropriate class. In the large sticks and circles network of Figure 1, nearly all data is public by definition of the single, large interface to each system. By developing appropriate methods for each class, we can begin to hide data within its class.

For example, consider the development of a firing solution for a given threat missile. In the large sticks and circles network, the firing solution uses public data that is visible to all other systems. Because the data is public and the network connects each system to all other systems, it is difficult for software designers to understand the impact on system behavior as it is not readily apparent what system functionality is dependent on the public data.

On the other hand, we can determine the data requirements for the development of the firing solution in the Weapon class in Figure 6, and understand that the software developers should hide that data within the Weapon class. While this data hiding may be more difficult in procedural software, the public data issue is more readily apparent in the class views of the system-of-systems than in the large sticks and circles network diagram.

Proc. Monterey Workshop: Radical Innovations of Software and Systems Eng. in the Future, US Army Research Office (Venice, Italy, Oct. 2002).

Summary. By applying UML and OOD techniques to the system-of-systems development, we can glean a great deal more insight into the system-of-systems requirements definition and allocation issues than the traditional sticks and circles diagrams so often used to depict these large, complex systems. By developing a class diagram with abstract classes for the major components of the system-of-systems, we can reason about the class diagram in our attempt to develop subclasses to which we can begin to allocate requirements and analyze system capabilities and limitations. Additionally, we can identify message requirements and message flow in our attempt to reduce coupling in the system-of-systems by developing requirements for simplified interfaces between the components. Finally, we can reassign methods to increase the cohesion of the components and we can hide data within a class to minimize the negative impacts of future modifications to either the system functionality or the data.

These aforementioned benefits of applying these UML and OOD techniques cannot be derived from the traditional views of system-of-systems designs. While software designers encounter other problems in system-of-systems designs, we believe that software developers can more easily reason about the system-of-systems requirements and associated allocation, thereby improving the system-of-systems designs by employing the techniques previously outlined in this discussion.

References

[1] Greenfield, Michael A., "Mission Success Starts With Safety," presentation to 19th International System Safety Conference, Huntsville, Alabama, September 11, 2001.

[2] Riel, Arthur J., *Object-Oriented Design Heuristics,* Reading, Massachusetts, Addison-Wesley, 1996.